# THE DESIGN OF A DISTRIBUTED COMPUTING SYSTEM

Marsha D. Hopwood, Donald C. Loomis,
Lawrence A. Rowe

The Distributed Computing System Project
Department of Information and Computer Science
University of California
Irvine, California 92664
June 1973

This work was supported by the National Science Foundation under Grant GJ-1045.



TECHNICAL REPORT #25 - JUNE 1973

## ABSTRACT

This paper describes a design for a computing system structure whose implementation should provide reliable, fail-soft service at relatively low cost. The design relies heavily on modularity and dynamic reconfigurability (in combination referred to as distribution) to achieve reliability. Low cost is maintained by using standard hardware and software components, each of which handles a portion of the workload. When a component fails, its workload is assumed by the remaining components. The evolution of a particular network architecture for this computing system, i.e., a network utilizing a communication loop with hardware components connected to and distributed about the ring, is described.

## INTRODUCTION

This paper describes the design of a distributed computing system (DCS), a system structure whose implementation should provide reliable, fail-soft service at relatively low cost. The system should be capable of providing high quality service to a large class of users with small to medium sized interactive tasks.

Reliability, in either hardware or software, is usually expensive. This paper characterizes reliability, indicates what techniques can be used to achieve this reliability, and shows how the system design decisions reconcile the conflicting goals of attaining reliability and keeping costs low. Some design alternatives which were rejected are also indicated and a brief description of how a prototype distributed computing system is being implemented is given.

## RELIABILITY

Attempts to attain reliability in the DCS concentrate on minimizing the consequences of system failure. In most computing systems, failure of a system component causes a significant disruption of service to all users. The DCS design attempts to minimize the impact of a component failure by localizing its effect to a small number of users. While the user computations actually utilizing a component

when it fails may be irrecoverable, other users need not be directly affected. If the component malfunction prevents its further use and the interrupted activities are restarted using the remaining components, all users will experience a degradation of response.

Reliability considerations encompass both hardware and software. In the discussion which follows both hardware and software faults are referred to as errors.

Reliability is defined in terms of the internal failure and error detection and correction capabilities of the computing system. In particular, there are six categories of performance, given in order of decreasing reliability. These are:

1. No errors occur.
2. Errors occur and are detected. Complete recovery is achieved with no loss of hardware or software.
3. Errors occur and are detected. Recovery is achieved with no loss of functional capability. Certain hardware or software components may become unavailable and a lowered system capacity may result.
4. Errors occur and are detected. Recovery is achieved but there is some loss of functional capability.

5. Errors occur but are detected externally, e.g., by a user of the system.

6. Errors occur and are detected neither internally nor externally.

While it is desirable that no errors occur, it is unrealistic to expect a system to be error-free. Therefore, the DCS design to achieve reliability attempts to minimize the probability of undetected errors and to maximize the possibilities for recovery from errors (fail-soft behavior).

The techniques considered for attaining reliability can be divided into two classes: techniques for error detection and techniques for recovery.

Error detection mechanisms are either system checks for errors or unexpected events calling attention to errors. Possible checks for errors include periodically running diagnostic tests while the system is operating, comparing results achieved in parallel, recognizing logical inconsistencies, and using error detection codes. Unexpected events which call attention to errors include interrupts, detection of "impossible" conditions, and communications from users.

If errors can be detected, techniques to minimize their effect can be considered. There are three categories of mechanisms to improve reliability: use of highly reliable

components, redundancy, and separation of components.

The use of highly reliable components is an error prevention technique which provides a straightforward solution to chronic errors. One can achieve a lower error rate by replacing a component which repeatedly fails by one of higher reliability.

Redundancy is used to mean either total or partial duplication of components. Hardware redundancy extends from the duplication of circuits in critical components to the complete duplication of the hardware system. All the redundant hardware may serve as an integral part of the computing system, may operate in parallel on the same tasks, or may be idle standbys to be used only when a failure occurs. Software redundancy extends from duplication of critical processes to the complete duplication of the software system and also includes redundant encoding of information. As with hardware, the workload may be distributed among all the components, the components may operate in parallel, or some components may be idle standbys.

Separation of components is a mechanism for logically and, where possible, physically isolating components. This is often called modularity. Separation, or partitioning, is significant to both error detection and error recovery. It

makes the determination of the source of an error easier since functions are more clearly delimited than in a nonmodular system. Without redundancy, separation of components allows error recovery only with loss of some functional capability. Many kinds of errors can disable particular components, but unless these components are critical, the entire system need not be disabled.

Some combination of redundancy and separation seems likely to be useful in attaining reliability while keeping costs low. Combinations of these two techniques are referred to as distribution. A distributed system can continue operation without loss of functional capability when redundant components fail.

## PRELIMINARY DESIGN DECISIONS

Many otherwise suitable designs to attain a high level of reliability in a computing system have to be discarded because they are costly. Several basic decisions have been made in the DCS design to allow reconciliation of the often conflicting goals of high reliability and low cost. These basic decisions serve as guidelines in the approach to the system design, but are not necessarily rigidly adhered to in all circumstances.

The first of these decisions is a commitment to use

readily available components as much as possible. Major deviations from this are likely to be very expensive. An example of a component which is not readily available is an ultra-reliable central processing unit. For many applications the increased reliability which can be attained with such hardware is not likely to be commensurate with the extra cost. Although certain applications with critical performance requirements, such as process control and air defense, may require nonstandard hardware and can provide justification for extra cost, there are many applications which do not. The handling of these less critical applications, for which performance requirements are flexible, degradation of performance is acceptable, and failure is not a major catastrophe, is the goal of this system.

A second basic decision is to restrict the way in which redundancy is employed in the computing system. To attain the maximum system capacity for a given cost, redundant components must increase potential system capacity, as well as improve reliability. The use of components as standbys or as parallel units limits the potential system capacity to a fraction of that which would otherwise be available when all components are operating properly.

A third basic decision is to limit the size of the

design and implementation of the system software. While the size of a software system is not a measure of its ultimate reliability, the smaller the software system is, the sooner it is likely to behave reliably. To reconcile reliability and low cost, software efforts should be kept small.

The basic design decisions lead to some particular decisions determining the structure of the hardware and software. These are described in the next two sections.

## Hardware Decisions

The first major design decision is to use several central processors in the DCS, rather than one. At a fixed cost, it is expected that the reliability of a computing system with several independent central processors will be greater than that of a system with a single central processor.

Three configurations can be considered. The first of these is a single central processor (or tightly coupled multiprocessor) system. Such a computing system can provide a wider range of services, particularly those having a relatively large hardware requirement, than a system whose computing capacity is not centralized. On the other hand, such a system is likely to be more susceptible to failure than a decentralized one, since most failures can disable the entire system.

Another configuration has several central processors whose total cost is roughly equivalent to the cost of the processor in the single processor system. Each of the processsors in the configuration is made the focus of a separate computing system. Such a system is likely to be more reliable than a single processor system, since the failure of one processor does not have to affect the other processors, but may have a narrower range of services. In particular, users whose needs are large in relation to the system capacity, e.g., users whose programs require large amounts of memory, cannot be served by such a system.

The third configuration, and the one adopted for the DCS, has several central processor/memory units which are loosely coupled to form a network. At a fixed cost this configuration is likely to be nearly as reliable as several separate systems and to provide functional capability closer to that of a single processor system.

An appropriately designed network organization can benefit from many of the advantages of the large single system as well as many of the advantages of smaller separate systems. They include:

1. Functional capability is retained when components fail. Except when the storage medium holding the user's data fails, he need not be aware of the

failure and will continue to have all of the system's functions available to him.

2. Processor specialization becomes feasible. This can lead to a reduction in the complexity of the software for the entire system, in addition to providing more efficient service.
3. Dynamic load balancing becomes possible. Activity can be allocated to the system components to balance the work load.
4. Peripherals can be distributed in an advantageous way. A full complement of peripherals is not necessary for each processor as might be necessary in separate systems. Part of the capacity of each machine need not be devoted to handling peripherals. Functionally specialized processors can be devoted to controlling peripherals.

One disadvantage of a network organization is that the cost of interconnection may be high. The network connection interfaces in the DCS have a few simple functions to perform and thus can be comparatively simple and inexpensive devices. This is in contrast to the network connection interface, the Interface Message Processor (IMP) in the ARPA Computer Network [12].

reliable components. Second, an aceptable level of performance capacity is achieved by using multiple copies of smaller computing components (e.g., processors) which are produced at low cost in high volume. Third, redundant components do not stand by idly waiting for a failure, but are available for use at all times.

A second cost, reconfiguration cost, is perhaps even more significant in the design of a computing system. If the cost of change can be kept small, a computing system can evolve to meet changing user requirements and changing availability of system components. Because the system consists of relatively small modules, its size can be changed incrementally by adding or removing components.

## DESIGN ALTERNATIVES FOR A DISTRIBUTED COMPUTING SYSTEM

The preliminary decisions indicate the general direction of the DCS design. Once these decisions were made, several possible organizations were available. This section looks at the design alternatives which were considered and indicates why particular ones were chosen.

### Hardware Decisions

The hardware design decisions are primarily concerned with the details of network organization. These include the manner in which the network components are connected and how

they communicate with each other [10].

Network Geography. Although the purpose in designing a system with modular components has been to increase reliability, the modularity makes it possible to separate components geographically if desired. All components may be contained in the same room, spread among a number of rooms (e.g., laboratories) in a small area, or distributed over a large geographical region. It was decided not to restrict the DCS design to any of these but to be adaptable to all.

Connection of Computing Components. The next topic considered is the connection of the computing components. There are at least five possible kinds of connections. These are:

1. direct connection (fully connected network)--each component connected to every other component via a two-way communication line;
2. specialized connection--connections between components as required by estimates of traffic;
3. switching center connection (star)--each component directly connected to a switching center which controls the routing of information to the components;
4. open-ended line (bus)--each component connected at some point on a two-way communication line; and

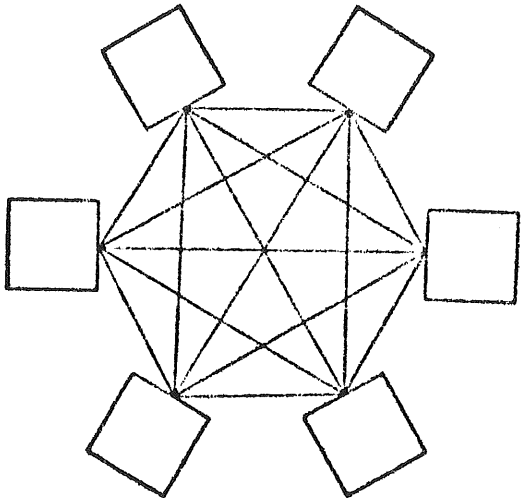
5. loop (ring)--each component connected at some point on a closed one-way communication line.

Figure 1 illustrates the possibilities for a system with six computing components.

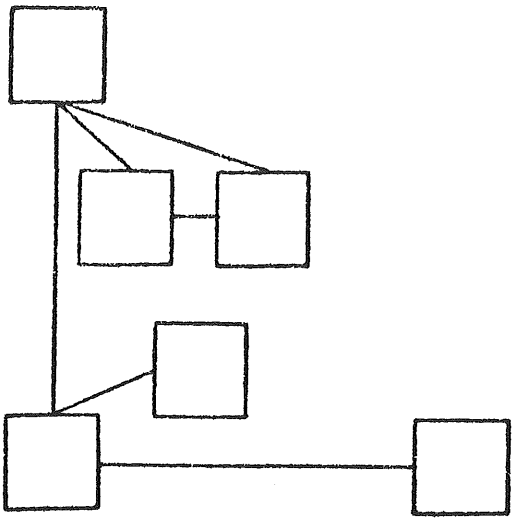
The ring organization was chosen over the other alternatives because it is inexpensive, new components are easy and inexpensive to add, and simple communication protocols are possible. Of particular importance in the simplification of message communication is the elimination of the need for a separate, explicit acknowledgment from the receiver of a message. This is possible because a message sent by a processor continues around the ring until it returns to the sender and, as the messages circulates, acknowledgment information can be appended to it.

A ring is more vulnerable than direct connection, since any break in the ring prevents all pairs of components from conversing. This vulnerability is reduced by adding secondary connections, or links, which skip connection points, thereby making it possible to exclude a malfunctioning primary section of the ring. This is illustrated in Figure 2.

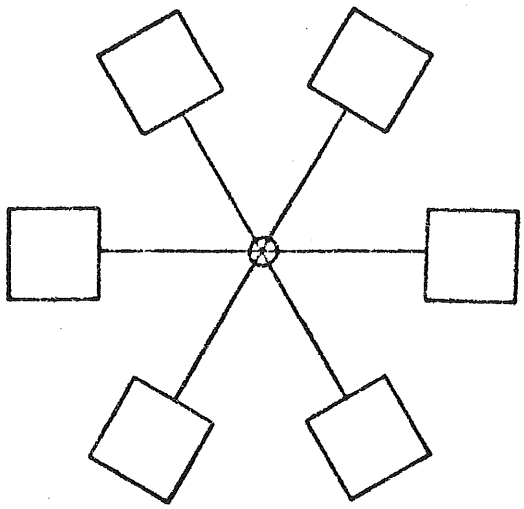
Communication Ring Control. Ring or loop type systems have been investigated at Bell Laboratories [7,11], IBM [14], and Collins [2]. They all have a loop supervisor



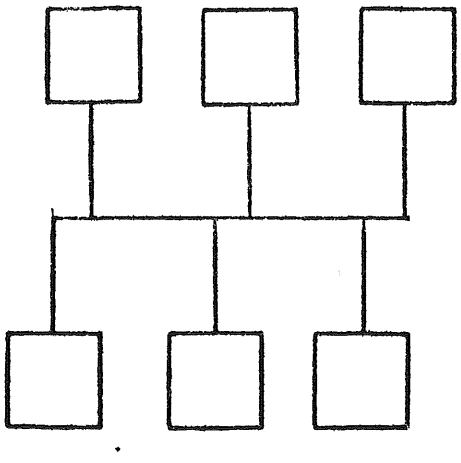
(1) Direct Connection



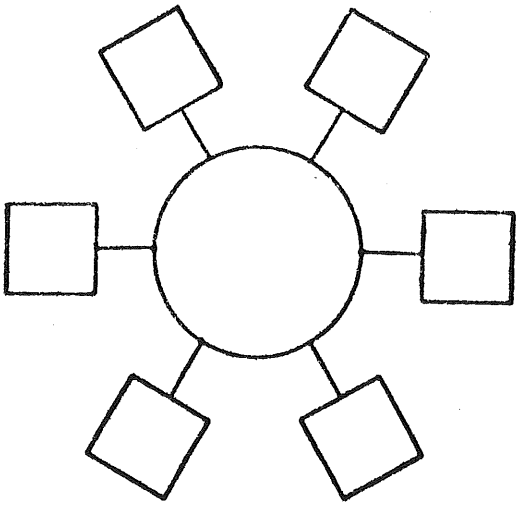
(2) Specialized Connection



(3) Switching Center Connection

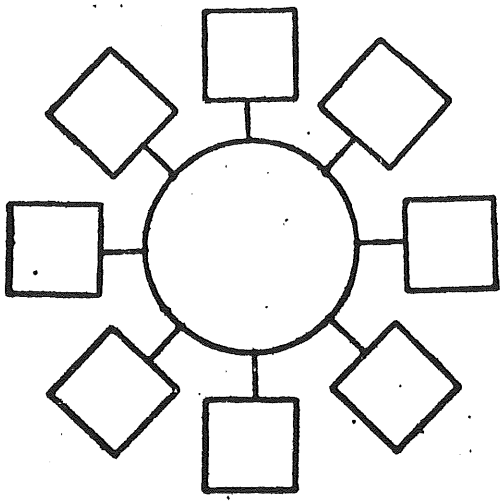


(4) Open-Ended Line Connection

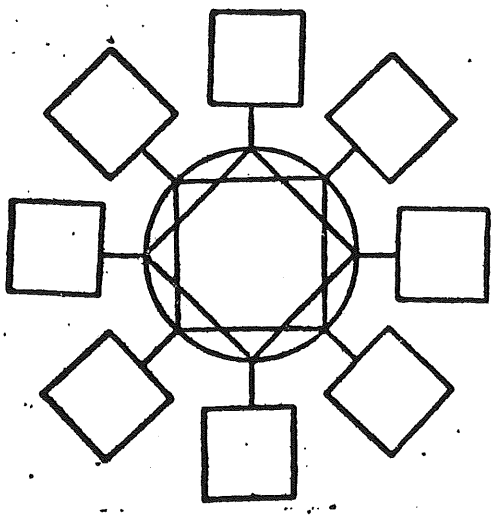


(5) Loop Connection

**Figure 1: Network Connection Alternatives.**



(a) Ring with Primary Links Only



(b) Ring with Primary and Secondary Links

Figure 2: Communication Ring with and without Secondary Links.

which is critical for proper operation of the ring. In order to attain high reliability, a ring which has a single loop supervisor has been rejected. Instead a completely symmetric ring with each node having an identical role in controlling the activities of the ring has been chosen. This distribution of control in the system's communication ring means that communications are not vulnerable to failure of a critical system component.

Relaying of Information. The next topic includes several considerations relating to the relaying of information around the ring. The first consideration is the alternative relaying methods. These are a store-and-forward scheme, in which an entire message is accepted before any part of it is relayed on to the next component, and a direct transmission scheme, in which messages are not stored before relaying. The store-and-forward method facilitates traffic leveling if alternative paths are available. The direct transmission method has several advantages. Messages get passed on more quickly and thus get to their destinations more quickly. In particular, the fact that there is essentially no delay at each of the relaying points makes this network organization viable even when components are distributed over a large geographical region. The direct transmission method also eliminates the need for a

significant amount of memory in the relaying mechanisms and simplifies coordinating the activities of these mechanisms. For these reasons direct relaying of messages has been chosen.

Another consideration is the rate at which messages are transmitted. Either the system components must respond at the rate of the ring, or a means for using only part of the ring bandwidth must be provided. Forcing the components to respond at ring speed has been chosen. This is simpler to implement since no fancy protocols are necessary. This has the disadvantage that buffers may be needed for those components which cannot send and/or receive at ring speed.

The role of the computing components (processors) in the handling of messages is also a concern. Either a processor is involved in handling all messages on the network or it is involved in only the messages sent to and from it. The second alternative has been chosen because to do otherwise could seriously affect the integrity of the computing system. A failure in any processor could disrupt all communication in the system. Also, the handling of all messages on the network by each processor could use a sizeable portion of the total computing resource.

Communications Protocols. The communications protocols present another set of alternatives. There are three which

were considered: control passing, lazy Susan, and delayed relaying. With control passing, only one relaying mechanism, or ring interface, is authorized to place a message onto the ring at any given time. All other ring interfaces can only relay messages. When the authorized ring interface is through transmitting, it will transmit a special sequence of bits known as a control token. Any other interface which then wishes to place a message on the ring replaces the control token with a message followed by a control token. If no ring interface needs to transmit, the control token will circulate around the ring.

The lazy Susan mechanism depends on a fixed message size and the division of the communication ring into trays of slots big enough to hold a message. If the time it takes for a message to go around the ring and return to its origin is sufficiently long, it may happen that by the time a message of fixed size is completely transmitted, the beginning of the message has only gotten 1/n of the way around the ring. The distance around the ring measured in bits is then n times the size of the message. The interval taken up by the message as well as the n-1 intervals of equal size can be viewed as message trays circulating around the ring. Any time an empty tray passes a ring interface, the interface may place a message in it.

The third mechanism, delayed relaying, allows the ring interface to delay incoming messages until it has completed the transmission of its own message. Of these three protocols, control passing has been selected. Analysis has shown [9,10] that it is simpler to implement, less seriously affected by errors, easier to recover from error, and can often provide better performance than the other protocols.

Network Devices. The last of the hardware alternatives to consider is the kind of devices which can be connected to a ring interface. One approach is to connect only processors to ring interfaces and connect all other devices to processors. The other alternative is to connect functional units, e.g., individual terminals or peripheral device-controller units, to ring interfaces. This has the advantage of decoupling access to various devices from processors. Also, since processors and peripherals are changing and their future characteristics are not clear, it is not desirable to fix the design to a unit which is likely to change.

## Software Decisions

There are several organizational decisions which were made about the software. These relate to structure and control.

System Software Structure. The processes which comprise the system software can be either hierarchical or nonhierarchical in organization. A hierarchical organization creates dependencies among the processes which can adversely affect reliability. For this reason a nonhierarchical organization [8] has been chosen.

Distribution or Centralization of Software. The next software issue relates to where services are to be provided. By centralization we mean that all services needed by a process running on a particular processor must be present on that processor. Distribution requires that only a few basic services be present on the processor. Other services may be performed on any processor. Distribution is the approach chosen.

Distribution or Centralization of Control. The final software issue relates to how the network is to be controlled. By centralization we mean that there will be a controlling process or supervisor which runs on one of the processors and that, in general, control of facilities will be concentrated in specific locations. Distribution or decentralization, on the other hand, allows control functions like scheduling and resource allocation to be performed by processes running on any processor. This is the approach adopted.

User Process Structure. The process structure which a user can create is also of concern. The kinds of structures users can create have been left open rather than have the system impose a structure. Thus the user can create a nonhierarchical structure similar to that found in the system software. A more rigid structure is also possible at the option of the user.

## A DISTRIBUTED COMPUTING SYSTEM

This section describes certain aspects of a prototype DCS which has evolved from the design considerations. The emphasis in this description is on how interactions are to be carried out in the network.

The DCS hardware system is a collection of computing system components connected to a digital communication ring by ring interfaces. The communication ring serves as a unidirectional information path and the ring interfaces assist in information routing. Figure 3 shows a DCS configuration with six processors.

The DCS is process oriented, that is, all activities are carried out by processes. Processes interact by sending and receiving messages. Messages are addressed to processes by name, rather than by physical hardware address. A message from one process addressed to another process is

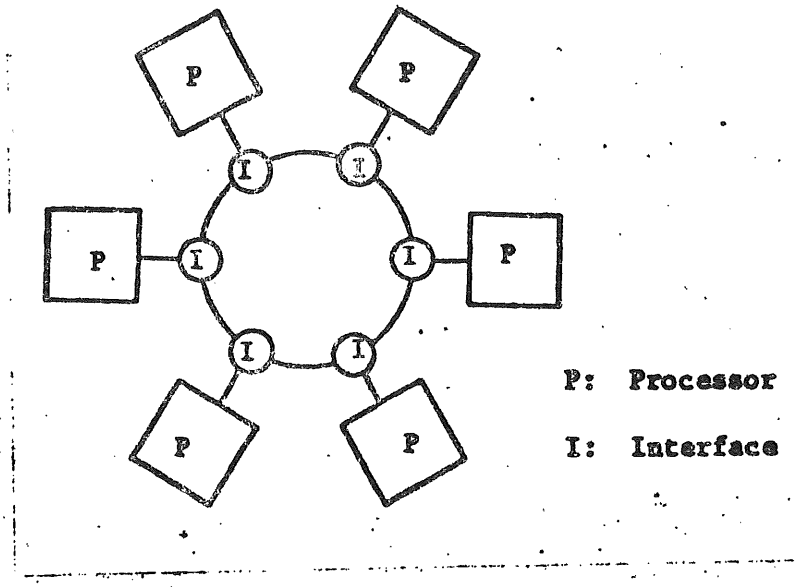



Figure 3: A Distributed Computing System with Six Processors.

placed onto the ring. As the message arrives at each ring interface, the interface compares the destination process name with its list of all processes active in the attached component. If the destination process name is present, the interface attempts to copy the message into the component memory. Whether the process is present or not, the interface allows the message to travel on to the next interface on the ring. The message continues around the ring until it arrives at the interface for the processor in which the sending process resides. This interface removes the message from the ring.

## System Services

Each processor on the ring has a resident software system called the nucleus. The nucleus provides facilities for scheduling processes and transmitting and receiving messages. Other system functions, such as resource allocation, device input-output, and file system services [5], are provided by processes executing in the DCS. Because the nucleus is the only software unit bound to a particular machine, these other system services may be executing in any machine in the ring and are accessed from user processes by sending and receiving messages. A process requesting service does not need to know where in the DCS the service process resides, because messages are addressed

to processes rather than processors.

## Protection

The functions one needs to protect a system--isolation and controlled access [1,3]--are explicitly included in the DCS design. In addition to the standard machine-level protection mechanisms, the DCS employs network-level protection. Because the DCS is process-oriented and process access is done by sending and receiving messages, protection is achieved by insuring the integrity of messages. In particular, protection is achieved by insuring that the sending process name placed in a message by the resident nucleus is actually the name of the process sending the message. If forgery of the sending process name has been prevented, a receiving process can discriminate in the services it provides depending on the source of the request.

## Failure Detection and Recovery

Failures in the DCS are detected when a communications problem arises, when an interrupt indicating an error occurs, or when a regular observable process action fails to occur. In the hardware there are three possibilities for failure [10]: a failure in the ring, a failure in a ring interface, and a failure in a component connected to a ring interface. Error detection facilities included in the

communications protocols are the primary means for detecting hardware failures such as intermittent transmission errors, failure of a ring interface, or complete interruption of a transmission. Periodic tests for failure of ring interfaces are also performed. If these facilities detect failures that are due to a faulty section of the ring or a faulty ring interface, secondary links (see Figure 2) can be used to logically exclude the malfunctioning part from the system. Processor/memory failures are recognized through hardware error detection circuits or through unexpected conditions detected by the software. Software errors almost always result in unexpected conditions.

Most failures in a DCS hardware or software component can be classified as either nucleus failures or process failures [13]. The failure of a nucleus is tantamount to a processor failure, since without the nucleus a processor cannot function in the DCS. The failure of a process, either a system or user process, is somewhat less serious than a nucleus failure, since its scope is likely to be smaller.

A nucleus failure is detected when a process fails to accept a message. (Accepting a message is a service performed by a nucleus.) This failure is recognized when a message is sent and one of two failure indicators is

returned as the status of the transmission attempt. One of the failure indicators signals that either a ring interface failure has occurred or the destination process no longer exists. The other failure indicator signals that either the input message buffers for the processor in which the destination process resides are full or the software nucleus in that processor has failed. Special processes, called status checkers, distinguish the various failure possibilities and initiate recovery actions when necessary. If a nucleus has failed, a remote restart process causes a bootstrap sequence to be activated in the failed processor. After appropriate reinitialization, a new copy of the nucleus is transmitted to the failed processor and the initiators of the processes previously executing in the processor are notified. If a nucleus fails repeatedly, a processor failure is assumed and action is taken to exclude that component from the system.

A process failure is detected when an interrupt indicating an error occurs or when a regular, observable action fails to occur. After a failure is recognized, the nucleus of the processor in which the failed process resides initiates communication with a status checker. The status checker then takes some action depending on the type of process which has failed. These actions include saving a

copy of the process environment, initiating a test process, initiating a new copy of the failed process, taking no explicit action until told to do so by an external source, and terminating the process.

Although the DCS design is intended to minimize the possibilities for failure, failures will nonetheless occur. The distribution of hardware, software, and system control makes it possible to employ relatively simple mechanisms to detect and recover from errors. Through the use of these mechanisms the effects of failures are minimized and fail-soft behavior is attained.

## CONCLUSION

In this paper we have given a rationale and design for a computing system to provide reliable, fail soft service at relatively low cost to a large class of users with modest requirements. The attempt to reconcile reliability and low cost led to the design of a computing system with a network architecture. The salient feature of this structure is the distribution of hardware, software, and control among the components of the network. This distribution is facilitated by the use of communication by name rather than address among the processes of the system. Because the system is both redundant and modular, it is relatively immune to total

failure and, in those instances where error or failure does occur, exhibits fail-soft behavior.

## ACKNOWLEDGMENTS

The authors wish to thank their colleagues on the DCS project for their assistance in preparing this paper and to acknowledge their contributions to the ideas presented here. In particular, we would like to acknowledge Professor David Farber and Professor Julian Feldman, the principal investigators who guided the system design, Frank Heinrich and Ken Larson, who made major contributions to the design of the software system, and Gregory Hopwood, who provided special assistance in implementing many of the ideas presented here.

## REFERENCES


1. Brinch Hansen, P. "The Nucleus of a Multiprogramming System." Comm. ACM 13, 4 (April 1970), pp. 238-241,250.

2. Collins Radio Company. C-System Working Papers. Cedar Rapids, Iowa.

3. Denning, P. J. "Third Generation Computer Systems." Computing Surveys 3, 4 (Dec. 1971), pp. 175-216.

4. Farber, D. J., J. Feldman, F. R. Heinrich, M. D. Hopwood, K. C. Larson, D. C. Loomis, and L. A. Rowe. "The Distributed Computing System." Proc. Seventh Annual IEEE Computer Society International Conference, (Feb. 1973), pp. 31-34.

5. Farber, D. J. and F. R. Heinrich. "The Structure of a Distributed Computer System--The File System." Proc. International Conference on Computer Communications, (Oct. 1972), pp. 364-370.

6. Farber, D. J. and K. Larson. "The Structure of a Distributed Computer System--The Communications System." Proc. Symposium on Computer-Communications Networks and Teletraffic, Microwave Research Institute of Polytechnic Institute of Brooklyn, (April 1972), pp. 21-27.

7. Farmer, W. D. and E. E. Newhall. "An Experimental Distributed Switching System to Handle High-Speed Aperiodic Computer Traffic." Proc. ACM Symposium on Problems in the Optimization of Data Communications Systems, (Oct. 1969).

8. Gord, E. P. and M. D. Hopwood. "Nonhierarchical Process Structure in a Decentralized Computing Environment." Technical Report #32, Department of Information and Computer Science, University of California, Irvine, California, (June 1973).

9. Heinrich, F. R. "Some Error Detection Schemes to Supplement the Cyclic Redundancy Check in the DCS Ring Transmission." Distributed Computing System Project Technical Memo #65, Department of Information and Computer Science, University of California, Irvine, California, (March 1973).

10. Loomis, D. C. "Ring Communication Protocols." Technical Report #26, Department of Information and Computer Science, University of California, Irvine, California, (Jan. 1973).

11. Pierce, J. R., C. H. Coker, and W. J. Kropfl. "An Experiment in Addressed Block Data Transmission Around a Loop." IEEE International Convention Digest, (March 1971), pp. 222-223.

12. Roberts, L. G. and B. D. Wessler. "Computer Network Development to Achieve Resource Sharing." Proc. AFIPS 1970 SJCC, Vol. 36, pp. 543-549.

13. Rowe, L. A., M. D. Hopwood, and D. J. Farber. "Software Methods for Achieving Fail-Soft Behavior in the Distributed Computing System." 1973 IEEE Symposium on Computer Software Reliability, (April 30, May 1-2, 1973), pp. 7-11.

14. Steward, E. A. "A Loop Transmission System." Proc. 1970 International Conference on Communications, Institute of Electrical and Electronics Engineers, (June 1970), pp. (36-1) - (36-9).